Order Notation and Estimating Complexity

We have looked at a few number of algorithms in class thus far. However, we have not looked at how to judge the efficiency or speed of an algorithm, which is one of the goals of this class.

We will use order notation to approximate two things about algorithms: how much time they take, and how much memory (space) they use.

The first thing to realize is that it will be nearly impossible to exactly figure out how much time an algorithm will take to execute on a particular computer. Each algorithm instruction gets translated into more small machine instructions, each of which take various amounts of time to execute on different computers. Also, we want to judge algorithms independent of their implementation. Thus, rather than figuring out an algorithm's exact running time, we will only want an approximation. The type of approximation we will be looking for is a Big-O approximation.

We will assume that each statement and each comparison in C takes some constant amount of time.

Also, most of the problems we will look at will have an input size. (For example, in sorting, the size of the input is the number of numbers to be sorted.) The time and space used by an algorithm will typically be a function of this input size. (The input size will typically be referred to as n.)

Big-Oh

Since we can't usually determine an exact number of steps an algorithm will take we'll be happy to make the two following simplifications in counting the number of steps an algorithm takes:

1) Eliminate any term whose contribution to the total ceases to be significant as n becomes large.

2) Eliminate constant factors.

Thus, if we happen to count that the number of steps a algorithm takes is $4n^2+3n - 5$, then we will

1) ignore 3n-5 because that accounts for a small number of steps as n gets large

2) Eliminate the constant factor of 4 in front of the n^2 term.

In doing so, we will conclude that the algorithm takes $O(n^2)$ steps.

In CS2, you will be introduced to the actual definition of Big-Oh. This is a simplification of the actual definition that is useful for most practical situations.

How does Order Notation Help Us Evaluate Algorithms?

Since we've determined that it may be too difficult to count up the exact number of steps an algorithm will take, we only want to be able to approximate an upper bound for the number of steps, within a constant factor. Hence, rather than saying that an algorithm will run $n^2 + n$ steps, we will be content to say that it runs $O(n^2)$ steps, since $n^2 + n = O(n^2)$.

One of the classic case studies in algorithms, (and usually the first taught to students), is the sorting problem. You may have informally seen this problem in COP 3223, but in this class we'll analyze the problem in far more detail and review even algorithms to sort a list of elements that you'll probably be sick of them by the end of it! We will use this problem as an introduction to algorithm analysis.